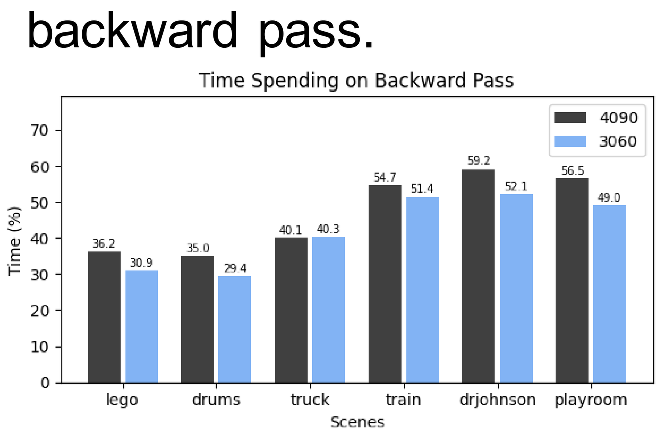
Atomic Aggregation on 3D Gaussian Splatting

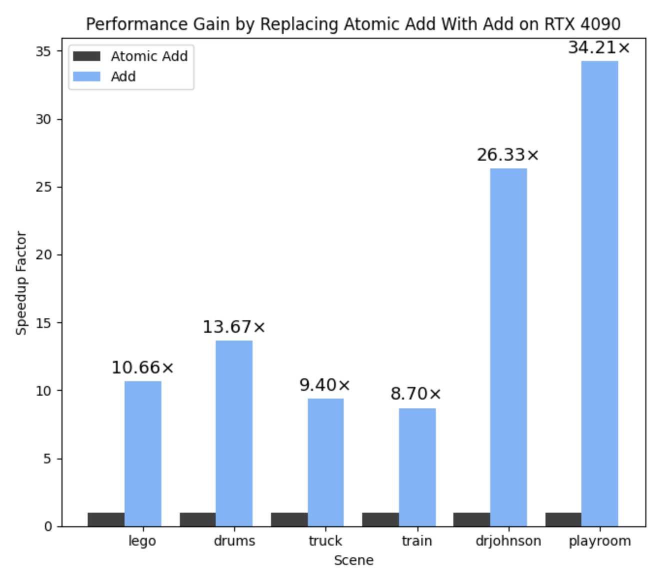
Fan Chen, Xin Peng, Keyi Zhang

Master of Science in Applied Computing, University of Toronto

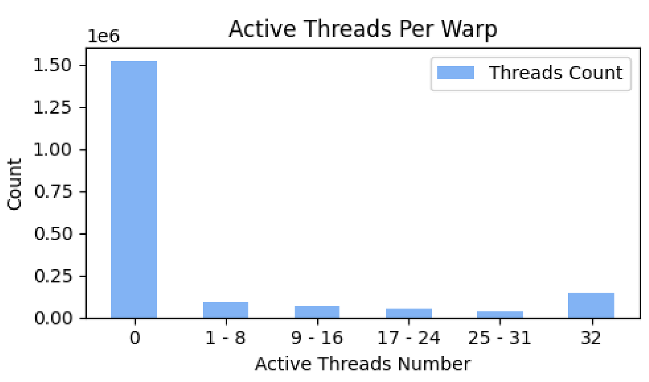
Motivation

3D Gaussian Splatting[1] achieves real-time rendering by leveraging GPU rasterization pipeline. However, the training of these models remains a computationally demanding process. By an in-depth performance analysis of 3DGS using Nvidia profiler[2], we identified the **atomic updates as a significant bottleneck** at the gradient computation step in backward pass.





Goal: Enhance the training efficiency by consolidating atomic instructions within the backward pass.



Key Observations:
(1) Threads within the same warp exclusively update identical memory locations.
(2) Atomic updates are solely performed by a subset of threads within a warp.

Key Contributions

- Conducted an exhaustive performance analysis on the training pipeline of 3DGS and discern atomic updates as a pivotal bottleneck.
- Introduced a software approach that uses warp-level reduction to reduce the number of atomic updates.
- Evaluated our approach on 3DGS application and demonstrate significant speed up.

Related Work

Atomic Processing in GPU:

- Individual threads perform atomic updates on specific data or memory locations to maintain data integrity.
- Ensures correct and conflict-free modifications in a parallel computing environment where multiple threads may simultaneously access the same memory location.

Warp Reduction:

```
#define FULL_MASK 0xffffffff
for (int offset = 16; offset > 0; offset /= 2)
    val += __shfl_down_sync(FULL_MASK, val, offset)
```

- Using the __shfl_down_sync primitive enables fast and direct data exchange between thread registers, which is more efficient than using shared memory[4]. This method can be used to accumulate results in a specific thread.

References

[1] B. Kerbl, G. Kopanas, T. Leimk ̈uhler, and G. Drettakis, “3d gaussian splatting for real-time radiance field rendering,” ACM Transactions on Graphics, vol. 42, no. 4, July 2023.
[2] “Nvidia nsight systems,” https://developer.nvidia.com/nsight-systems
[3] “Nvidia nsight compute,” https://developer.nvidia.com/nsight-compute, accessed: 2023-11-20
[4] “Using cuda warp-level primitives,” https://developer.nvidia.com/blog/using-cuda-warp-level-primitives/, accessed: 2023-11-20.

Method

```
function GRADCOMPUTATION(prims_per_thread)
    tid ← thread_idx                                  ▷ Thread corr. to pixel
    for p : primitives[tid] do                        ▷ Iterate
        skip ← false
        if COND1 then
            skip ← true                               ▷ Mark inactive status
        end if
        ...
        if COND2 then
            skip ← true                               ▷ Mark inactive status
        end if
        ...
        if SKIP then
            grad_x1 ← 0
            grad_x2 ← 0
            grad_x3 ← 0
        end if
        active_count ← __popc(__ballot_sync(__activemask(), !skip))
        if !ACTIVE_COUNT then
            continue;                                 ▷ warp doesn't participate
        end if
        REDUCTION(grad_t x1)
        REDUCTION(grad_t x2)
        REDUCTION(grad_t x3)
        if LANE_ID == 0 then
            ATOMICADD(p.grad_x1, grad_t x1)
            ATOMICADD(p.grad_x2, grad_t x2)
            ATOMICADD(p.grad_x3, grad_t x3)
        end if
    end for
end function
```

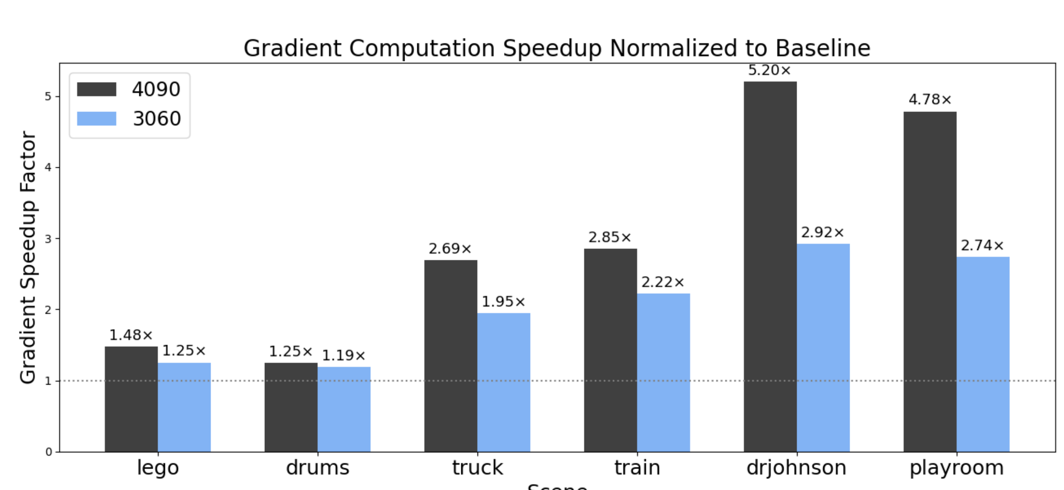
- Introduce `skip` to manage threads that do not participate in atomic updates, ensuring all threads can synchronize during warp-level reduction by assigning inactive threads a value of 0.
- Based on Observation (2), if no threads are active, the loop skips unnecessary reduction and atomic addition operations, streamlining the process.
- Based on Observation (1), perform a warp-level reduction to consolidate gradient calculations into a single value per gradient component within the warp, which is then atomically added to the global memory by a single thread.

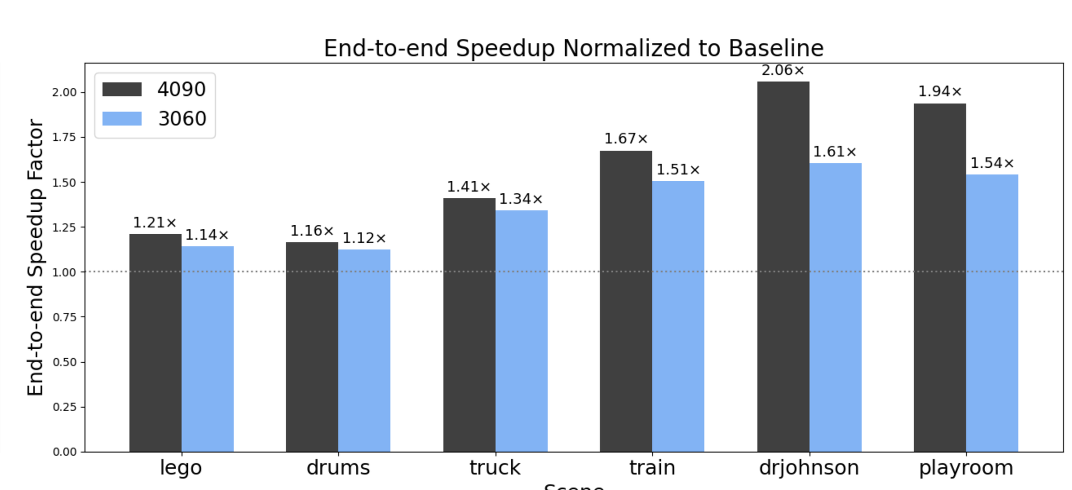
Experimental Results

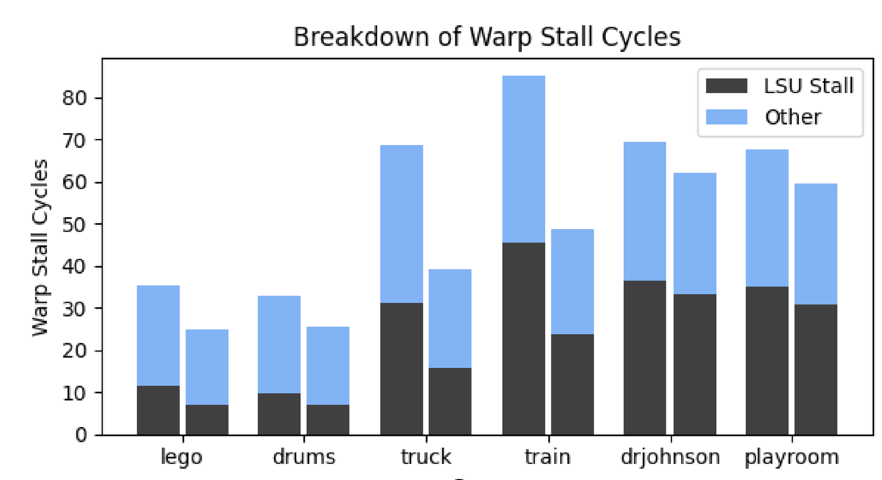
- PSNR results for six datasets, acquired during a 5-minute training period on both the RTX 4090 and RTX 3060, employing both the baseline and our approach.

	4090		3060	
	Baseline	**Ours**	**Baseline**	**Ours**
lego	35.793	35.682	33.708	34.264
drums	29.990	30.124	28.856	29.034
playroom	32.843	34.975	29.988	30.195
drjohnson	29.549	32.160	28.025	29.076
truck	24.779	25.851	23.723	24.170
train	23.591	23.638	19.393	21.371

- Left shows the normalized speedup specifically for the gradient computation using Nvidia Nsight Compute[3]. Right shows the normalized speedup for the end-to-end runtime, including the forward pass using Nvidia Nsight System[2].







- The breakdown of the number of cycles a warp is stalled per instruction on the NVIDIA RTX 4090 and 3060 GPUs. Left-top is the result of baseline, left-bottom is the result of our approach.
- Below is the load store unit utilization for both the baseline and our proposed approach